클래스 없는 자바스크립트

클래스와 this 없이 자바스크립트를 개발하는 법.

- 객체지향에서 중요한 통찰은, 프로그램끼리 메시지로 통신하는 것.
- 상속은 클래스 사이에 강한 결합을 유발함.
- 이 장에서 class 도 this 도 사용하지 않고 프로그램을 작성하는 법에 대해 서술함.
- 상속 대신 `composition` 을 사용하여 중복을 제거함.
- (이게 지금 리액트 hook의 방향 아닌가? 하는 생각이 드네요)

생성자

- 생성자란 객체를 반환하는 함수. 팩토리와 비슷한 방법으로 만들 수 있음.

```
function counter(){
    let counter = 0;

    function up(){
        counter +=1;
        return counter;
    }

    function down(){
        counter-=1;
        return counter;
    }
    return Object.freeze({up,down});
}
```

- 1장부터 계속 얘기해왔던 방식으로 만들어진 카운터 코드.
- 이제보니 hook 과 유사한 구조네요.
- 객체는 동결되어서 손상되거나 오염되지 않고, `up`, `down`을 통해서만 상태 변경이 발생합니다.

생성자의 초기화

- 생성자를 만들 때 객체 리터럴 하나만 전달받는 생성자로 만들자

```
function counter(spec){
const{name,color,type} = spec;// 받지않은 값은 undefined 로 되어서 무시.
}
```

- 인자 전달 순서가 상관 없음
- 코드 수정하지 않아도 인자 추가 가능
- 가독성이 증가
- 맞는것같음. 인자 넘길때 이렇게 넘기는거 괜찮은듯!

합성

- 상속보다 합성. 너무나 유명한 말!
- 사실상 `hook`을 사용하는 리액트 개발자는 어쩔 수 없이 합성을 맨날 사용하고 있음.

```
function example(spec){
    let {member} = spec;
    const reuse = other_example() // 훅 안에서 다른 훅 불러오는거랑 똑같음....
    const method(){
        // 이 안에서 reuse 도 member도 사용가능.
    }
    return Object.freeze({method})
}
```

- 읽다보니 진짜 리액트의 발전 방향이 더글라스가 추구하는 방향하고 일치한다는 생각이 좀 듦...
- 이렇게 사용하면 메모리가 프로토타입 방법보다 많이 드는건 팩트
- 근데 지금 머신 메모리가 16기가 이러니까 완전 티끌같은것.

17장의 후기

- 타입스크립트를 쓰면서는 리액트 외에는 클래스를 주로 사용하고 있었는데, 읽다보니 굳이 그래야 하나 생각도 들기 시작함.
- `funtiuon` 에도 인터페이스가 있으니 다형성을 활용하는데에도 문제가 안되지 않을까? 생각이 들고... 한번 클래스 없이 짜봐야겠다는 생각이 들었음.

꼬리재귀

- 꼬리재귀는 함수형 프로그래밍의 단짝같은 존재라고 생각함.
- 함수형 관련 책에서 매번 설명해줌.
- 먼저 꼬리호출이란 함수가 함수 호출 결과를 바로 반환하는것.

```
function tail(arg1){
    return function tali2(arg2,arg3){
        return arg2(arg1(arg3))
    }
}
```

- 이를 이해하려면 함수가 불릴때 어떤 일이 발생하는지 알아야 함.
- 함수는 불릴때 `내가 돌아갈 장소` 를 저장함.

- 자바스크립트의 경우 예전에 한번 말한적이 있는 활성객체에 이를 저장함.
- 그래서 `tail2` 함수가 리턴됐을 때 `tail1` 내부로 돌아오라고 저장해 줄 것.
- `arg2` 함수가 리턴됐을 때 `tail2` 안으로 돌아오라고 저장해 줄 것.
- 하지만 실제로는 그렇지 않음.
- `arg2` 함수는 리턴됐을 때 `tail2` 가 아니라 `tail1` 으로 바로 돌아감.
- 즉, 별도의 메모리를 사용하지 않고 리턴 포인트를 최적화 시킴.
- (모든 언어가 이를 지원하는게 아님! 언어차원에서 이를 지원해줘야 함.)
- 즉 리턴포인트를 유지할 필요가 없게 코드를 작성해주면 엄청난 최적화를 할 수 있음.
- 이게 바로 `꼬리재귀 최적화`.

- 책에는 나오지 않지만... 우리가 흔히 사용하는 `for` 같은 제어문은 제어문을 한 번 돌때마다 스택을 클리어해주는 기능이 있음
- 꼬리재귀 최적화를 이용하면 마치 `for`를 사용하는 정도로 재귀문을 최적화 할 수 있음.

최적화 된 경우와 아닌 경우는 얼마나 다른가?

자바스크립트는 함수가 불리면 다음과 같은 동작을 함

- 함수의 매개변수와 변수를 저장할 충분한 활성객체(신)를 생성함
- 호출된 함수 객체에 대한 참조를 활성객체(신)에 저장
- 전달받은 인자를 활성객체(신)에 저장
- 함수 호출 명령어의 다음 명령어를 활성객체(신)의 다음 명령어 필드에 넣어줌
- 활성객체(신) caller 필드에 현재 활성객체를 넣어줌.
- 현재 활성 객체를 활성객체(신) 으로 업데이트함.
- 호출된 함수를 실행함.
  -> 우리가 다 잘 알고있는 내용.

그렇담? 최적화가 되면?

- 현재 활성객체가 충분히 크다면, 현재 활성 객체를 새로운 활성객체로 활용함.
- 나머지는 그대로.
  -> 하지만 대부분의 경우 활성객체는 충분히 큼.
- 메모리 할당과 가비지 컬랙션 시간이 많이 감소함.
- 재귀호출을 반복문만큼 빠르게 할 수 있음
- 스택 오버플로우가 발생하지 않음.

꼬리재귀의 조건

- 좋은건 알겠는데... 꼬리재귀의 조건을 잘 알아야 함.
- 함수가 반환하는 값이 바로 반환되어야 꼬리최적화가 발생함.

```
return (
    typeof any === 'function'? any() : undefined
)
- 꼬리재귀임
```

```
return any() + 1
- 꼬리재귀가 아님. 리턴값으로 뭔가를 해야하기 때문. 즉, 리턴값을 사용하기 때문에
- 리턴 포인트를 변경할 수가 없음
```

```
any()
return;
```
- 꼬리재귀가 아님. 리턴값이 함수의 반환값이 아니기 때문. undefined 가 리턴될것.

```
const return_value = any()
return return_value
```
- 꼬리재귀가 아님. 리턴값이 any()의 결과긴 하지만, 꼬리가 아니어서 최적화되지 않음.(요새 컴파일러가 얼마나 똑똑한데 이정도는 해주지...싶긴 함)

- 일반적인 재귀함수는 꼬리재귀가 아님

```
function fact(n){
    if(n<2) return 1;
    return n * fact(n-1);
}
```

- 왜인지 맞춰보세요!

```
function fact(n,result = 1){
    if(n<2) return result;
    return fact(n-1, n* result);
}
```

- 이건 꼬리 최적화가 되었을까요?

마무리

- 클래스 없는 자바스크립트하고 꼬리재귀에 대해 알아보았습니다
- 클래스 없는 자바스크립트는 보면서 현재의 리액트와 많이 닮았다고 생각이 들었습니다.
- 꼬리재귀는 알고 있었던 내용이지만, 실제 개발할 때 재귀를 잘 사용하지 않아서 가물가물했던 내용인데 정리가 잘 되어서 좋네요.
- 재귀를 짤 때 의식하면서 꼬리 최적화를 하면서 사용해야겠다는 생각을 하며 마무리.